

Paper Interpreter

2025年4月22日 17:27

Title (English & Japanese)

Attention Is All You Need

「Attentionこそが全て」

Journal Name & Publication Year

NIPS 2017 (Neural Information Processing Systems 第31回会議)

First and Last Authors (English only)

First Author: Ashish Vaswani

Last Author: Illia Polosukhin

First Affiliations (English only)

Google Brain

Abstract

本論文では、従来のリカレントや畳み込みネットワークに依存する系列変換モデルとは異なり、Attentionメカニズムのみに基づく新しいネットワークアーキテクチャ「Transformer」を提案する。Transformerは並列処理が可能で、従来手法より高速に学習可能。WMT 2014の英独翻訳タスクでBLEUスコア28.4を達成し、既存の最良手法（アンサンブル含む）を2.0以上上回った。英仏翻訳タスクでもBLEU 41.8を記録。さらに構文解析など他タスクへの応用でも良好な汎化性能を示した。

Background

従来の機械翻訳モデルでは、RNNやCNNをベースにしたエンコーダ・デコーダ構造が主流であり、Attention機構は補助的に使用されていたが、本研究はそれらを完全に排除し、Attentionのみによって系列変換を行う初のモデルを提案。

Methods

Transformerは自己注意（Self-Attention）と位置ごとのFeed-Forward層のみで構成され、リカレント成分を完全に除去。各トークン間の依存関係は、Scaled Dot-Product AttentionおよびMulti-Head Attention（8ヘッド）で処理される。位置情報はSin/Cos関数に基づくPositional Encodingを用いて埋め込む。エンコーダ・デコーダ共に6層の構成（baseモデル）を採用し、各層に残差接続とLayer Normalizationを追加。

Results

- WMT 2014英独翻訳：BLEU 28.4（Transformer bigモデル、従来最良より+2.0以上）
- WMT 2014英仏翻訳：BLEU 41.8（3.5日、8 GPU、シングルモデル）
- 構文解析（WSJ Section 23）：F1スコア 92.7（semi-supervised setting）

Transformer baseモデルは100Kステップ（約12時間）で既存のNMTアンサンブルを上回る性能を達成。

Discussion

自己注意層は、計算コストを抑えつつ長距離依存関係を学習しやすく、並列化が容易な構造である点が大きな利点。Attention Headは構文や照応など異なる言語的機能を自律的に学習。位置エンコーディングにはSin/Cos方式が有効で、学習済み埋め込みと同等の性能を発揮した。

Novelty compared to previous studies

Attentionのみで系列変換を行う点が最大の革新。従来はRNN/CNNが必要とされていたが、本研究は完全にAttentionに置き換えることに成功し、翻訳精度と学習効率の両面で既存手法を凌駕。自己注意による短経路学習と高速化が両立された初の成功例。

Limitations

自己注意はメモリ消費量が入力長に対して二乗スケールする（ $O(n^2)$ ）ため、極端に長い系列には不向き。これを解決するためのローカル注意や制限付き注意が今後の研究課題とされている。

Potential Applications

自然言語処理全般（翻訳、要約、QA、文解析）に加えて、画像、音声、動画など他モダリティへの拡張も示唆。構文解析など構造的制約のあるタスクでも高性能を実現しており、汎用的な深層学習アーキテクチャとしての展開が期待される。

Attention Is All You Need

Ashish Vaswani*
Google Brain
avaswani@google.com

Noam Shazeer*
Google Brain
noam@google.com

Niki Parmar*
Google Research
nikip@google.com

Jakob Uszkoreit*
Google Research
usz@google.com

Llion Jones* Google
Research llion@google.com

Aidan N. Gomez* †
University of Toronto
aidan@cs.toronto.edu

Łukasz Kaiser*
Google Brain
lukaszkaiser@google.com

Illia Polosukhin* ‡
illia.polosukhin@gmail.com

Abstract

優勢な配列伝達モデルは、エンコーダーとデコーダーを含む複雑なリカレントまたは畳み込みニューラルネットワークに基づいている。最も性能の良いモデルは、注意メカニズムを介してエンコーダとデコーダも接続する。我々は、再帰と畳み込みを完全に排除し、注意メカニズムのみに基づく新しいシンプルなネットワークアーキテクチャ、Transformerを提案する。2つの機械翻訳タスクの実験から、これらのモデルは並列化可能であり、学習時間が大幅に短縮される一方で、品質が優れていることが示された。我々のモデルはWMT 2014英独翻訳タスクで28.4 BLEUを達成し、アンサンブルを含む既存の最良結果を2 BLEU以上上回った。WMT 2014の英仏翻訳タスクにおいて、我々のモデルは8つのGPUで3.5日間学習した後、41.8という新しい単一モデルの最新BLEUスコアを確立した。Transformerは、大規模な学習データと限られた学習データの両方で英語の構文解析にうまく適用することで、他のタスクにうまく汎化することを示す。

↑ エンコーダ/デコーダー型のRNN or CNN型のNNを用いた時系列モデルが主流であり、アテンションを追加した性能の良いモデルが登場している

⇒ 本論文では、RNNとCNNを除外し、アテンションのみに基づくアーキテクチャ「Transformer」を考案する

⇒ 本手法は並列化可能であり、かつ高精度

1. Introduction

2025年4月23日 4:29

1 Introduction

リカレントニューラルネットワーク、特に長期短期記憶型[13]とゲートドリカレント[7]ニューラルネットワークは、言語モデリングや機械翻訳などのシーケンスモデリングやトランスダクション問題における最先端のアプローチとして確固たる地位を築いている[35, 2, 5]。その後、リカレント言語モデルやエンコーダ・デコーダのアーキテクチャの限界を押し広げるために、数多くの努力が続けられている[38, 24, 15]。

リカレントモデルは通常、入力シーケンスと出力シーケンスのシンボル位置に沿った計算を因数分解する。計算時間のステップに位置を合わせると、前の隠れ状態 h_{t-1} と位置 t の入力の関数として、隠れ状態 h_t のシーケンスを生成する。このような本質的に逐次的な性質は、学習例内での並列化を妨げ、メモリ制約が例間のバッチングを制限するため、シーケンス長が長くなると重要になる。最近の研究では、因数分解のトリック[21]や条件付き計算[32]によって計算効率の大幅な改善を達成し、後者の場合のモデル性能も改善されている。しかし、逐次計算の基本的な制約は残っている。

注意メカニズムは、様々なタスクにおける説得力のあるシーケンスモデリングとトランスダクションモデルに不可欠な要素となっており、入力シーケンスや出力シーケンスにおける距離に関係なく依存関係をモデリングすることができます[2, 19]。しかし、ごく一部の例[27]を除いて、このような注意メカニズムはリカレントネットワークと組み合わせて使用される。

この研究では、再帰を避け、代わりに入力と出力の間のグローバルな依存関係を描くために、注意メカニズムに完全に依存するモデルアーキテクチャであるTransformerを提案する。Transformerは大幅に並列化を可能にし、8台のP100 GPUでわずか12時間学習した後、翻訳品質の新しい状態に到達することができる。

↑ RNN、特にLSTMとGRUは時系列モデリングで最先端の手法であり、エンコーダー/デコーダー型のアーキテクチャの限界向上のために、様々な工夫が研究されている

⇒ RNNは、入力/出力シーケンスの位置に沿った計算を逐次処理行うが、並列化できないことによる処理時間の増加とメモリ制約が課題（最近の研究では計算効率の改善もあったが、逐次処理がまだボトルネック）

⇒ アテンションを使うことで、シーケンスの距離に関係なく依存関係をモデリングできるようになったが、一般的にRNNと組み合わせて使われる

⇒ そこで、本論文ではアテンションのみで構成されるTransformerを提案

⇒ これにより、並列化と翻訳精度の向上を実現

2. 背景

2025年4月23日 4:34

2 Background

逐次計算を減らすという目標は、Extended Neural GPU [16]、ByteNet [18]、ConvS2S [9]の基礎にもなっており、これらはすべて畳み込みニューラルネットワークを基本構成要素として使用し、すべての入出力位置に対して隠れ表現を並列に計算する。これらのモデルにおいて、任意の2つの入出力位置からの信号を関連付けるために必要な演算数は、位置間の距離に応じて、ConvS2Sでは線形に、ByteNetでは対数的に増加する。このため、離れた位置間の依存関係を学習することが難しくなる[12]。Transformerでは、これは一定の演算数に削減されるが、注意で重み付けされた位置を平均化することによる有効解像度の低下という代償を払うことになる。この効果は、3.2節で説明したように、マルチヘッド注意で打ち消す。

自己注意(intra-attention)とは、シーケンスの表現を計算するために、1つのシーケンスの異なる位置を関連付ける注意メカニズムである。自己注意は、読解、抽象的要約、テキスト含意、タスクに依存しない文表現の学習など、様々なタスクで成功裏に使用されている[4, 27, 28, 22]。

エンドツーエンドのメモリネットワークは、配列に沿った再帰ではなく、再帰的な注意メカニズムに基づいており、単純な言語の質問応答や言語モデリングタスクで良好な性能を発揮することが示されている[34]。

しかし、我々の知る限り、Transformerは、配列整列RNNや畳み込みを使用せずに、入力と出力の表現を計算するために完全に自己注意に依存する最初のトランスダクションモデルである。以下のセクションでは、Transformerについて説明し、自己注意を動機付け、[17, 18]や[9]のようなモデルに対する優位性について議論する。

↑ 逐次処理を減らす努力は、CNNを用いた先行研究で行われてきたが、任意の2つの入出力の関連付けの計算は、2つの距離が遠くなるほど増加するため、学習が困難になる

⇒ Transformerでは、一定の演算数に減らすために、アテンションで重みづけされた位置を平均化することをするが、これにより解像度が低下する(マルチヘッドアテンションにより、この欠点は相殺可能)

⇒ 自己注意は、1つのシーケンス中の位置を関連づけるアテンションであり、Transformerは自己注意のみに依存

この文章は、自己注意（Self-Attention）とTransformerの登場意義について語っている部分で、特に「逐次処理を減らす」ことが目標であるとしています。以下にポイントを押さえてわかりやすく解説します。

✓ 全体のテーマ

RNNのような「逐次計算」を減らし、並列計算を可能にするために、いくつかの新しいモデルが提案されてきた。その流れの中で、Transformerは自己注意のみで構成された最初のモデルとして登場し、特に注目された。

◆ 従来のモデル：並列化を目指した工夫

以下のモデルたちは、RNNに代わって並列処理可能な構造を模索してきた：

✓ Extended Neural GPU、ByteNet、ConvS2S

- いずれも **CNN（畳み込みニューラルネットワーク）** を使って逐次性を回避
- 全ての位置の隠れ状態を並列に計算可能（＝高速）

ただし、それぞれに限界もある：

モデル名	入出力位置間の関係性を学習するための演算量
ConvS2S	距離に対して 線形に増加
ByteNet	距離に対して 対数的に増加

→ つまり、離れた位置同士の依存関係を捉えるのが難しい（遠くなるほど情報の伝達が大変）

◆ Transformerの革新：演算量が「一定」

Transformerでは、どの位置間の関係も「一定の演算回数」で結びつけられる（全ての単語同士が直接つながる）

これは Self-Attention（自己注意） によって実現されている。

◆ 自己注意（Self-Attention）の定義と効果

自己注意とは「シーケンス内の異なる位置同士の関係性」を学習するための注意メカニズム

例：

「彼はリンゴを食べた。それは赤かった。」→ “それ” が “リンゴ” を指す、など

✓ 自己注意の利点：

- 各単語が文中のどの単語に注目すべきかを学習できる
- 並列に計算可能
- 応用が広い：読解、要約、文表現など多くのNLPタスクで有効（例：[4], [27], [28], [22]）

◆ Self-Attentionの限界：有効解像度の低下

ただし、注意をかけすぎて全体平均のようになると、焦点がボケてしまう（重要な部分が埋もれる）

この問題を解消するために使われるのが **Multi-Head Attention**（マルチヘッド注意）。

異なる視点から同時に注意を分散することで、「解像度の低下」を防ぎます（3.2節で詳述）。

◆ Transformerの特徴

Transformerは、RNNやCNNを一切使わず、自己注意だけで入出力の特徴表現を作る初のモデル

つまり：

- 完全に並列化可能
- 計算パスが短く、遠距離依存も容易に学習
- 多数のタスクでSOTA（最先端性能）を達成

✓ まとめ（要点）

ポイント	内容
🎯 目的	RNNのような逐次計算を減らし、並列化を実現すること
🔧 中間モデル	Extended Neural GPU、ByteNet、ConvS2S：CNNベースで試みたが限界あり
🌟 Transformer	自己注意だけで構成された初の完全Attention型モデル。並列処理可能で効率的
⚠️ 欠点	注意が分散しすぎると情報が埋もれる → マルチヘッドで解決
🏆 優位性	他モデルよりも「遠距離依存」を少ない演算で表現でき、強力なモデルとなった

3. モデル構造

2025年4月23日 4:39

3 Model Architecture

ほとんどの競合するニューラル配列変換モデルは、エンコーダ・デコーダ構造を持つ[5, 2, 35]。ここで、エンコーダは入力記号表現列 (x_1, \dots, x_n) を連続表現列 $z = (z_1, \dots, z_n)$ に写像する。 z が与えられ、デコーダはシンボルの出力列 (y_1, \dots, y_m) を1要素ずつ生成する。各ステップにおいて、モデルは自己回帰的であり[10]、以前に生成された記号を次の記号を生成する際の追加入力として消費する。

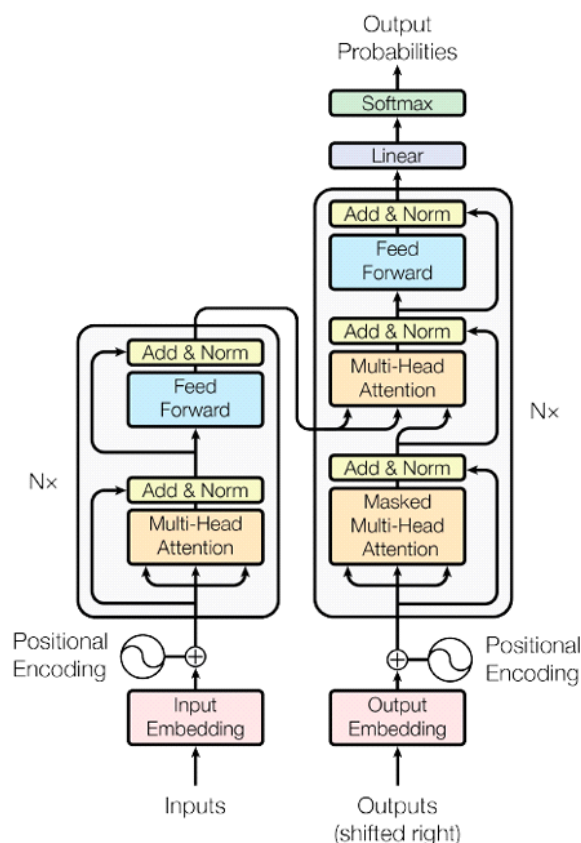


図1:Transformerモデルのアーキテクチャ。

Transformerは、図1の左半分と右半分にそれぞれ示すように、エンコーダーとデコーダーの両方に、スタックされた自己注意層とポイント単位の完全接続層を使用して、この全体的なアーキテクチャに従う。

↑ 一般的なSq2Sqモデルは、エンコーダーとデコーダーで構成される

⇒エンコーダー：入力表現列を連続表現列に変換

⇒デコーダー：連続表現列から出力列を1要素ずつ生成

⇒また、各ステップでモデルは過去に生成したものも参考に次のものを生成する(自己回帰的)

⇒Transformerでは、エンコーダーとデコーダーの両方に、スタックされたセルフアテンション層とMLPを接続した構成になっている

3.1 エンコーダとデコーダのスタック

エンコーダ: エンコーダは $N=6$ 個の同一レイヤーのスタックで構成される。各層は2つのサブレイヤーを持つ。1つ目はマルチヘッド自己アテンションメカニズムで、2つ目は単純な、位置的に完全接続されたフィードフォワードネットワークである。2つのサブレイヤーのそれぞれの周りに残差接続[11]を採用し、その後レイヤーの正規化[1]を行う。すなわち、各サブレイヤーの出力は $\text{LayerNorm}(x + \text{Sublayer}(x))$ であり、 $\text{Sublayer}(x)$ はサブレイヤー自身が実装する関数である。これらの残差接続を容易にするために、埋め込み層と同様にモデル内の全ての副層は $d_{\text{model}} = 512$ 次元の出力を生成する。

デコーダ: デコーダも $N=6$ 個の同一レイヤーのスタックで構成される。各エンコーダ層の2つのサブレイヤーに加えて、デコーダは第3のサブレイヤーを挿入し、エンコーダスタックの出力に対してマルチヘッドアテンションを実行する。エンコーダと同様に、各サブレイヤーの周囲に残差接続を採用し、レイヤーの正規化を行う。また、デコーダスタックの自己アテンションサブレイヤーを変更し、ポジションが後続のポジションにアテンションしないようにする。このマスキングは、出力埋め込みが i 位置オフセットされることと組み合わせられ、位置 i の予測が i より小さい位置の既知の出力にのみ依存することを保証する。

↑ エンコーダは以下①②で構成された構造を6回繰り返す

⇒①マルチヘッドセルフアテンション

②MLP

⇒二つとも残差接続を採用し、その後にレイヤーのノルム(トークンごとの平均0分散1の正規化)を行い、学習しやすくする

⇒デコーダーは①②に③のサブレイヤーを追加

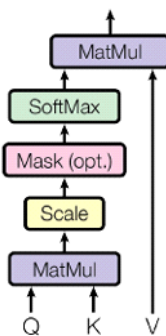
⇒また、セルフアテンションをmaskされたアテンションに変更し、位置 i の単語を生成するとき、

i 以降の生成単語の情報を見ないようにする

3.2 Attention

アテンション関数は、クエリとキーと値のペアのセットを出力にマッピングするものとして記述することができ、クエリ、キー、値、および出力はすべてベクトルである。

スケールドットプロダクトアテンション



Multi-Head Attention

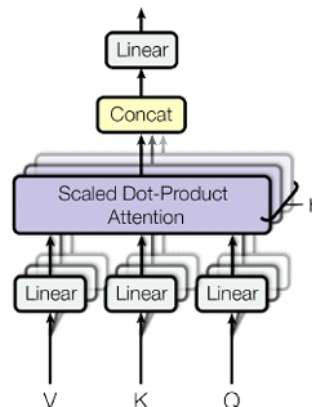


図2:(左)スケールドットプロダクトアテンション。(右)マルチヘッドアテンションは、並列に実行される複数のアテンションレイヤーから構成される。

出力は値の加重和として計算され、各値に割り当てられた重みは、対応するキーとクエリの互換性関数によって計算される。

↑ QとKで重みを作って、それをVに対して、加重平均してる

⇒マルチヘッドアテンションはQ,K,Vを分割して、並列計算してから最後に基に次元になるように結合してる

我々はこの特殊な注意を「スケールドドットプロダクト注意」と呼ぶ(図2)。入力は d_k 次元のクエリとキー、および d_v 次元の $\sqrt{\quad}$ 値からなる。全てのキーとクエリのドット積を計算し、それぞれを d_k で割り、ソフトマックス関数を適用して値の重みを求める。

実際には、クエリの集合を同時に行列 Q に詰め込んで注目関数を計算する。また、キーと値は行列 K と V にまとめて詰め込まれる。出力の行列を次のように計算する：

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (1)$$

最もよく使われる注意関数は、加法的注意[2]とドット積(乗法)注意の2つである。ドット積注目度は、 $\sqrt{1}$ のスケールリングファクターを除いて、我々のアルゴリズムと同じである。加法的注意は、 d_k を単一の隠れ層とするフィードフォワードネットワークを用いて互換性関数を計算する。この2つは理論的な複雑さでは似ているが、ドット積アテンションは高度に最適化された行列乗算コードを用いて実装できるため、実際にはより高速でスペース効率も高い。

d_k の値が小さい場合には、2つのメカニズムは同様の性能を示すが、 d_k の値が大きい場合には、加法的注意はスケールリングせずにドット積注意を上回る[3]。 d_k の値が大きいと、ドット積の大きさが大きくなり、ソフトマックス関数が極端に小さい勾配⁴を持つ領域に押し込まれるのではないかと推測される。この効果を打ち消すために、ドット積を $\sqrt{1}$ でスケールリングする。^d

↑ **QKで重みを作って、softmaxで確立にしている**

(d_k は学習しやすいようにしてる)

⇒それにVをかけてアテンションベクトルの生成

この画像は、**Transformerのコア技術である「Scaled Dot-Product Attention」**について解説している重要な部分です。
以下に、数式も含めてわかりやすく日本語で説明します。

✓ 概要：Scaled Dot-Product Attention とは？

Transformer における**注意機構（attention mechanism）**の基本的な形式であり、以下の3つを入力に持ちます：

- Q ：Query（照会）
- K ：Key（鍵）
- V ：Value（値）

目的は、QueryとKeyの内積から重み（attention）を計算して、Valueにそれをかけて出力を得るというものです。

🔴 数式 (1) の意味

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^{\top}}{\sqrt{d_k}} \right) V$$

記号	意味
QK^{\top}	QueryとKeyの内積（各クエリと各キーの類似度）
$\sqrt{d_k}$	Keyの次元数の平方根でスケーリング（正規化）
softmax	類似度を確率に変換（合計1の重みになる）
V	Valueに重みをかけて、情報を合成する

✅ なぜスケーリング（ $\frac{1}{\sqrt{d_k}}$ ）が必要？

特に d_k （Keyの次元数）が大きいと、内積の値が大きくなりすぎて、softmaxが「極端な分布」になる。

- 結果として、softmaxの勾配が極端に小さくなり（勾配消失）、
- 学習が進みにくくなる。

✅ 対策：スケーリングすることで、softmaxの値が安定し、学習しやすくなる。

📦 他のアテンション（加法型）との違い

- **加法アテンション（additive attention）**は、MLP（1層のFFNN）を使ってスコアを出す方式（Bahdanau attention）。
- **内積アテンション（dot-product attention）**は、単純なベクトル内積。
- 本手法（scaled dot-product attention）は、「内積アテンション」にスケーリングを加えたもの。

✅ なぜこれが使われるのか？

- 理論的には加法型と同じくらい複雑だが、
- 実装上は、行列積とsoftmaxで高速に処理できる。
- そのため、Transformerでは「Scaled Dot-Product Attention」が採用されている。

💬 一言まとめ

Scaled Dot-Product Attention は、「類似度（内積）をスケーリングしつつsoftmaxに通すことで、安定した注意重みを得て、Valueを重み付け合成する」手法であり、Transformerの要（かなめ）です。

図解や、Query-Key-Valueの意味の視覚的説明が必要でしたらお伝えください！

3.2.2 マルチヘッド注意

d_{model} -次元のキー、値、クエリで単一の注意関数を実行する代わりに、クエリ、キー、値をそれぞれ d_k 、 d_k 、 d_v 次元に異なる学習済み線形投影で h 回線形投影することが有益であることがわかった。

これらの投影されたクエリ、キー、値のそれぞれに対して、注意関数を並列に実行し、 d_v -次元の出力値を得る。これらを連結し、もう一度投影すると、図2に示すような最終的な値が得られる。

マルチヘッドアテンションにより、モデルは異なる位置の異なる表現部分空間からの情報に共同してアテンションすることができる。単一の注意ヘッドでは、平均化によってこれが阻害される。

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$
$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.

この研究では、 $h = 8$ 個の並列注意層(ヘッド)を採用する。これらそれぞれについて、 $d_k = d_v = d_{\text{model}} / h = 64$ とする。各ヘッドの次元が小さくなるため、総計算コストは全次元の単一ヘッド注意と同様である。

↑ QKVを分割して、重み自体も分割することで、いろんな重みを学習できる！

この画像は、**Transformerの「Multi-Head Attention (マルチヘッドアテンション)」**に関する説明部分です。以下に、数式・構造・直感を交えてわかりやすく解説します！

✓ 目的：なぜ「マルチヘッド」にするの？

単一のアテンション（＝シングルヘッド）だけでは、学習できる関係性の表現に限りがある。

マルチヘッドにすることで：

- 異なる視点（subspace）で同時にアテンションを計算できる
- 複数の「意味的なつながり」を並列に学習できる（例：構文的・意味的・距離的關係など）

💡 処理の流れ（まとめ）

1. 入力：Q, K, V（それぞれ d_{model} 次元）

- Queries, Keys, Values は元々すべて d_{model} 次元（例：512）

2. h 個のヘッドに分ける（ここでは $h = 8$ ）

- 各ヘッドごとに、異なる学習可能な行列 W_i^Q, W_i^K, W_i^V を使って次元変換：

$$QW_i^Q, KW_i^K, VW_i^V \quad (\text{それぞれ } \in \mathbb{R}^{d_{\text{model}} \times d_k} \text{ など})$$

- 通常 $d_k = d_v = d_{\text{model}} / h$ （例：64）

3. 各ヘッドで Scaled Dot-Product Attention を並列に計算：

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

4. 全ヘッドの出力を結合（Concat）

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

- 最後に再び W^O という線形層を通して、元の d_{model} に戻す

✔ 特徴とメリット

特徴	説明
並列処理	各ヘッドが同時に注意を計算できる（GPUに優しい）
表現力向上	異なる関係性を並行して学習できる
高速	各ヘッドは低次元なので、全体のコストはシングルヘッドと同程度

✔ 最後のポイント

実装上、各ヘッドは「次元を下げて処理」することで、**計算量を増やさずに情報量を増やす**ことができる。

例えば：

- $d_{\text{model}} = 512$
- $h = 8$
- 各ヘッドの $d_k = d_v = 64$

→ 全体の計算量は維持しつつ、多様な注意が実現できる。



一言でまとめると：

Multi-Head Attention = 小さな注意機構 × 複数 → 合体して強力な注意

異なる視点から情報に注意を向けることで、Transformerの表現力を飛躍的に向上させています。

重みを分割して学習した方が、いろんな重みを学習できるってこと？

はい！その理解、とても良いです。

簡単に言うと、**「重み（アテンション）を分割して学習することで、より多様な視点から情報を捉えられる」**というのが **Multi-Head Attention（マルチヘッドアテンション）** の本質です。

✔ もっと詳しく：なぜ分割（＝マルチヘッド）が良いのか？

● 単一ヘッド（Single-Head Attention）の場合：

- 全体の表現（例えば512次元）で1つのアテンションを計算する
- 1つの重み行列に全ての意味的关系を詰め込まなきゃいけない
- → 表現力に偏りや制限が出る

● マルチヘッド（Multi-Head Attention）の場合：

- 入力を複数の「サブ空間」に分割（例：512次元 → 64次元 × 8ヘッド）
- 各ヘッドごとに独立したアテンション重みを学習
- 各ヘッドが異なるタイプの関係性を学習可能！

🔍 例え：同じ文を8つの視点で読むようなイメージ

ヘッド1	文法的関係に注目（主語-述語）
ヘッド2	同義語・対義語の意味的距離に注目
ヘッド3	文中の繰り返しに注目
ヘッド4	時制や助詞に注目
...	...

→ それぞれの「目（= attention head）」が違うところを見ていて、全体を合体（concat）することで表現が多様で豊かになる！

✅ ポイントまとめ

質問	答え
「重みを分割したら、学べるが増えるの？」	✅ はい。ヘッドごとに独立した重みを持つので、多様な情報を同時に学習できます。
「学習の効率は下がらないの？」	❌ 下がりにません。むしろヘッドごとの次元を下けているので、計算コストは同等です（並列化もできる）
「結局何が得られるの？」	🔍 複数の意味的・構文的関係を同時に扱える表現力の高さです！

必要であれば、「マルチヘッドがどんな重みを学習するかの可視化例」も紹介できます！興味ありますか？

3.2.3 我々のモデルにおける注意の応用

Transformerは3つの異なる方法でマルチヘッドアテンションを使用します：

- エンコーダ・デコーダ注目層では、クエリは前のデコーダ層から、メモリキーと値はエンコーダの出力から来る。これにより、デコーダ内のすべての位置が、入力シーケンス内のすべての位置にわたってアテンションすることができる。これは、[38, 2, 9]のようなシーケンス間モデルにおける典型的なエンコーダ・デコーダの注意メカニズムを模倣したものである。
- エンコーダは自己注意層を含む。自己アテンション層では、キー、値、クエリはすべて同じ場所、この場合はエンコーダの前の層の出力から来る。エンコーダの各位置は、エンコーダの前層のすべての位置にアテンションできる。
- 同様に、デコーダの自己アテンション層は、デコーダの各位置が、その位置までのデコーダのすべての位置にアテンションすることを可能にする。自己回帰特性を維持するために、デコーダの左向きの情報フローを防ぐ必要がある。我々は、不正な接続に対応するソフトマックスの入力のすべての値をマスク（ $-\infty$ に設定）することで、スケールされたドット積注意の内部でこれを実装する。図2を参照。

はい、このセクション「3.2.3 我々のモデルにおける注意の応用」は、Transformer アーキテクチャの中で マルチヘッドアテンション（Multi-Head Attention）をどのように使い分けているかを詳しく説明しています。

以下、3つの使い方それぞれについて日本語でわかりやすく解説します。

✓ Transformerにおけるアテンションの3つの使い方

① エンコーダ・デコーダアテンション

デコーダがエンコーダ出力に対して注意を向ける

- クエリ（Query）：デコーダの出力（翻訳中の文）
- キー・バリュー（Key & Value）：エンコーダの出力（原文）

🗨 目的：

- デコーダが「翻訳中の現在の位置から、原文のどこを見れば良いか」を学習する
- 機械翻訳での典型的なアテンションの使い方（Seq2Seq + Attentionと同様）

② エンコーダの自己アテンション（Self-Attention）

エンコーダ内部で、入力文の全単語同士が注意を向け合う

- Query = Key = Value = エンコーダの出力（前の層）

🗨 目的：

- 入力文中の単語同士が文法や意味に基づいて相互に関連付ける
- たとえば、「the cat sat on the mat」の中で、「the」と「cat」が関係あると学ぶ

③ デコーダの自己アテンション

デコーダが「過去の単語にだけ注意」を向けて、自分の翻訳履歴を参照する

- Query = Key = Value = デコーダの出力（前の層）

🗨 特別な工夫：「マスキング（masking）」

- 翻訳時、未来の単語はまだ生成されていないので見てはいけない
- softmaxのスコアを計算する前に、未来の位置の値を マイナス無限大（ $-\infty$ ）に設定してマスク
- これにより、デコーダは「過去の単語だけを見て次の単語を予測」できるようになる（自己回帰性の維持）

✓ 図で表すと（※図2の内容）

rust

📄 コピーする

✎ 編集する

エンコーダ

- └ Self-Attention（全単語同士が注目）
- └ 出力 → デコーダへ渡る

デコーダ

- └ Masked Self-Attention（未来を見ない）
- └ Encoder-Decoder Attention（入力に注目）
- └ 出力 → 翻訳結果生成

🟢 まとめ

アテンションの種類	クエリ	キー / バリュース	目的
Self-Attention (Enc)	入力	入力	入力文内の関係把握
Masked Self-Attention (Dec)	翻訳中の文	自分の過去	自己回帰的に次単語を予測
Encoder-Decoder Attention	翻訳中の文	原文	翻訳時に原文のどこを参照するか学習

💬 一言まとめ

Transformerは「文中の関係」「翻訳履歴」「原文との対応」それぞれにアテンションを使い分けており、マルチヘッド注意の使いどころが非常に戦略的に設計されています。

必要があれば、図解付きでこの3種類のアテンションの流れを可視化できますよ！興味ありますか？

3.3 位置ごとのフィードフォワードネットワーク

注意のサブレイヤーに加えて、我々のエンコーダとデコーダの各レイヤーは、各位置に別々に同一に適用される完全接続のフィードフォワードネットワークを含んでいる。これは2つの線形変換とその間のReLU活性化からなる。

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \quad (2)$$

線形変換は異なる位置で同じであるが、層ごとに異なるパラメータを使用する。これを説明するもう一つの方法は、カーネルサイズ1の2つの畳み込みである。入力と出力の次元は $d_{\text{model}} = 512$ であり、内層の次元は $d_{\text{ff}} = 2048$ である。

↑ アテンションで作った埋め込みベクトルにReLUのFFを通す

⇒これが必要なのは後続研究で議論されている

3.4 埋め込みとソフトマックス

他の配列変換モデルと同様に、入力トークンと出力トークンを次元 d_{model} のベクトルに変換するために学習済み埋め込みを使用する。また、通常の学習済み線形変換とソフトマックス関数を用いて、デコーダ出力を予測される次のトークンの確率に変換する。我々のモデルでは、[30]と同様に、2つの埋め込み層とプリソフトマックス/線形変換の間で同じ重み行列を共有する。埋め込み層では、これらの重みに d_{model} を乗じる。

表1:異なるレイヤータイプにおける最大パス長、レイヤーごとの複雑さ、および最小逐次演算数。 n はシーケンス長、 d は表現次元、 k は畳み込みのカーネルサイズ、 r は制限された自己注意における近傍のサイズである。

Layer Type	Complexity per Layer	Sequential Operations	最大経路長
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(\log_k(n))$
自己アテンション(制限付き)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

↑ 単語をベクトルに埋め込む方法と、最終的に次の単語を予測するソフトマックスについて書いてる

3.5 位置エンコーディング

我々のモデルには再帰性も畳み込みもないので、モデルがシーケンスの順序を利用するためには、シーケンス内のトークンの相対位置または絶対位置に関する情報を注入しなければならない。この目的のために、エンコーダとデコーダのスタックの底にある入力埋め込みに「位置エンコーディング」を追加する。位置エンコーディングは埋め込みと同じ次元 d_{model} を持つので、両者を合計することができる。位置エンコーディングには、学習型と固定型の多くの選択肢がある[9]。

本研究では、異なる周波数のサイン関数とコサイン関数を使用する：

$$PE_{(pos, 2i)} = \sin(pos/10000^{2i/d_{model}})$$
$$PE_{(pos, 2i+1)} = \cos(pos/10000^{2i/d_{model}})$$

ここで、posは位置、iは次元である。すなわち、位置エンコーディングの各次元は正弦波に対応する。波長は 2π から $10000 - 2\pi$ までの幾何学的な進行を形成する。任意の固定オフセットkに対して、 PE_{pos+k} は PE_{pos} の一次関数として表現できるため、モデルが相対位置による出席を容易に学習できると仮定して、この関数を選択した。

また、代わりに学習済みの位置埋め込み[9]を使用する実験も行い、2つのバージョンでほぼ同じ結果が得られることがわかった(表3の行(E)参照)。正弦波バージョンを選んだのは、トレーニング中に遭遇したシーケンスよりも長いシーケンス長にモデルを外挿できる可能性があるからである。

↑ **Transformerでは単語の位置がわからなくなるので、周期関数を利用して位置ベクトルを作成し、元のベクトルに足し合わせた**

4. Why Self Attention

2025年4月23日 6:37

4 Why Self-Attention

このセクションでは、自己注意層の様々な側面を、ある可変長の記号表現列(x_1, \dots, x_n)を、 $x_i, z_i \in \mathbb{R}^d$ と等長の別のシーケンス(z_1, \dots, z_n)にマッピングするためによく使われるリカレント層と畳み込み層、例えば典型的なシーケンス変換エンコーダやデコーダの隠れ層などと比較する。自己注意を使う動機付けとして、3つの望ましい条件を考える。

1つはレイヤーごとの総計算量である。もう一つは、並列化できる計算量であり、必要とされる逐次演算の最小数で測定される。

3つ目は、ネットワーク内の長距離依存関係間のパスの長さである。長距離依存性の学習は、多くの配列変換タスクにおける重要な課題である。このような依存関係を学習する能力に影響を与える重要な要因の1つは、前方信号と後方信号がネットワーク内で通過しなければならない経路の長さである。入力配列と出力配列の任意の位置の組み合わせの間のこれらのパスが短ければ短いほど、長距離依存性の学習が容易になります[12]。したがって、異なる層のタイプで構成されるネットワークにおいて、任意の2つの入出力位置間の最大経路長も比較する。

表1にあるように、自己アテンション層は一定数の逐次実行操作で全ての位置を接続するのに対し、リカレント層は $O(n)$ の逐次実行操作を必要とする。計算量の点では、配列長 n が表現次元 d より小さい場合、自己注意層はリカレント層より高速であり、これは機械翻訳の最先端モデルで用いられる文表現、例えばワードピース[38]やバイトペア[31]表現で最もよく見られるケースである。非常に長いシーケンスを含むタスクの計算性能を向上させるために、自己注意を、それぞれの出力位置を中心とした入力シーケンスのサイズ r の近傍のみを考慮するように制限することができる。これにより、最大経路長は $O(n/r)$ に増加する。今後の研究で、このアプローチをさらに調査する予定である。

カーネル幅 $k < n$ の単一の畳み込み層は、入力と出力の位置のすべてのペアを接続しない。そのため、連続カーネルの場合は $O(n/k)$ 、拡張畳み込みの場合は $O(\log_k(n))$ の畳み込み層を積み重ねる必要があり[18]、ネットワーク内の任意の2つの位置間の最長パスの長さが長くなる。畳み込み層は一般にリカレント層より k 倍も高価である。しかし、分離可能な畳み込み[6]は、複雑さを大幅に減少させ、 $O(k - n - d + n - d^2)$ となる。しかし、 $k = n$ であっても、分離可能な畳み込みの複雑さは、我々のモデルで採用している自己注意層とポイントワイズフィードフォワード層の組み合わせに等しい。

副次的な利点として、自己アテンションはより解釈しやすいモデルをもたらす可能性がある。我々のモデルから注意の分布を検査し、付録で例を提示し、議論する。個々の注意の頭は明らかに異なるタスクを実行するように学習するだけでなく、多くは文の構文構造や意味構造に関連した振る舞いを示すようである。

このセクションは、自己注意（Self-Attention）層が他の典型的なシーケンス処理層（リカレント層や畳み込み層）と比べてどんな利点があるかを、数理的・構造的に分析しています。以下にポイントを絞って分かりやすく解説します。

✓ 比較の目的：Self-Attention vs RNN vs CNN

自己注意層（Self-Attention Layer）を、以下の観点から **RNN（リカレント）やCNN（畳み込み）**と比較：

1. 総計算量（Compute Complexity）
2. 並列性（Parallelizability）
3. 長距離依存関係の学習能力（Path Length）

✓ 1. 計算量と並列化

● Self-Attention の特徴

- 全単語が全単語に直接アテンションできる（全結合）
- → 1ステップで全位置に並列にアクセス可能
- → 計算量は $O(n^2 \cdot d)$ （ n = 長さ、 d = 次元）

● RNN（リカレント）の問題点

- 順番にしか処理できない（時間的に逐次）
- → 並列化できない
- → 計算量は $O(n \cdot d^2)$ でパス長も $O(n)$

● CNN（畳み込み）の中間的な特性

- カーネルサイズが限られているため、広い範囲をカバーするには多層化が必要
- 拡張畳み込み（dilated conv）などでも、長距離依存性のパス長は log スケール
- CNN 自体は並列処理可能だが、パス長が短くならない

✓ 2. 長距離依存のパス長

- パス長とは、ある入力位置と出力位置がどれだけ間接的に接続されるかの長さ（情報伝達のステップ数）
- 自己注意は1ステップで任意の位置同士が接続できる（パス長 = 1）
- RNNは順にしか伝わらない（パス長 = $O(n)$ ）
- CNNはカーネルの重ね数に応じてパス長が減る（例： $O(\log n)$ ）

✓ 3. 制限付き自己注意（制限範囲 r の例）

計算量を抑えるために、「自己注意を全体にではなく、各出力位置から近傍 r 個の範囲だけに限定する」手法も検討されている

- このとき最大パス長は $O(n/r)$ に増える
- → 精度と速度のトレードオフができる
- これは**後続研究（Longformer, Performerなど）**にもつながる発想

✓ 3. 制限付き自己注意（制限範囲 r の例）

計算量を抑えるために、「自己注意を全体にではなく、各出力位置から近傍 r 個の範囲だけに限定する」手法も検討されている

- このとき最大パス長は $O(n/r)$ に増える
- → 精度と速度のトレードオフができる
- これは**後続研究（Longformer, Performerなど）**にもつながる発想

✓ 計算コストの実際

- 通常の CNN は RNN より高コスト（カーネル幅 k に依存）
- ただし「分離可能な畳み込み（separable convolution）」にすると複雑さが大幅に減る
- $k = n$ のように広いカーネルでも、self-attention + FFN と同等になる

✓ 解釈性の高さもメリット

自己注意は、「どの単語がどの単語に注目しているか」を重みで可視化できる。

- 文法構造（例：主語と動詞）や意味構造（例：「それ」が「猫」に対応）などが明示的に見える
- 複数のアテンションヘッドは、それぞれ異なる言語的役割を自然に学習する

🧠 まとめ

比較軸	Self-Attention	RNN	CNN
並列性	✓ 高い（全並列）	✗ 逐次処理	✓ 高い
長距離依存パス長	1（最短）	$O(n)$	$O(\log n) \sim O(n/k)$
解釈性	✓ 可視化可能	✗	✗
計算量	$O(n^2 \cdot d)$	$O(n \cdot d^2)$	$O(n \cdot k \cdot d)$ （分離可能なら高速）

🔔 一言まとめ：

Self-Attentionは、並列処理できて長距離依存も捉えやすく、かつ解釈性が高いという特徴を持ち、RNNやCNNの限界を打破する構造としてTransformerの中心技術になった。